

A Parameterized Model Transformations Approach for Automating Middleware QoS Configurations in DRE Systems

Amogh Kavimandan, Aniruddha Gokhale
{amoghk,gokhale}@dre.vanderbilt.edu

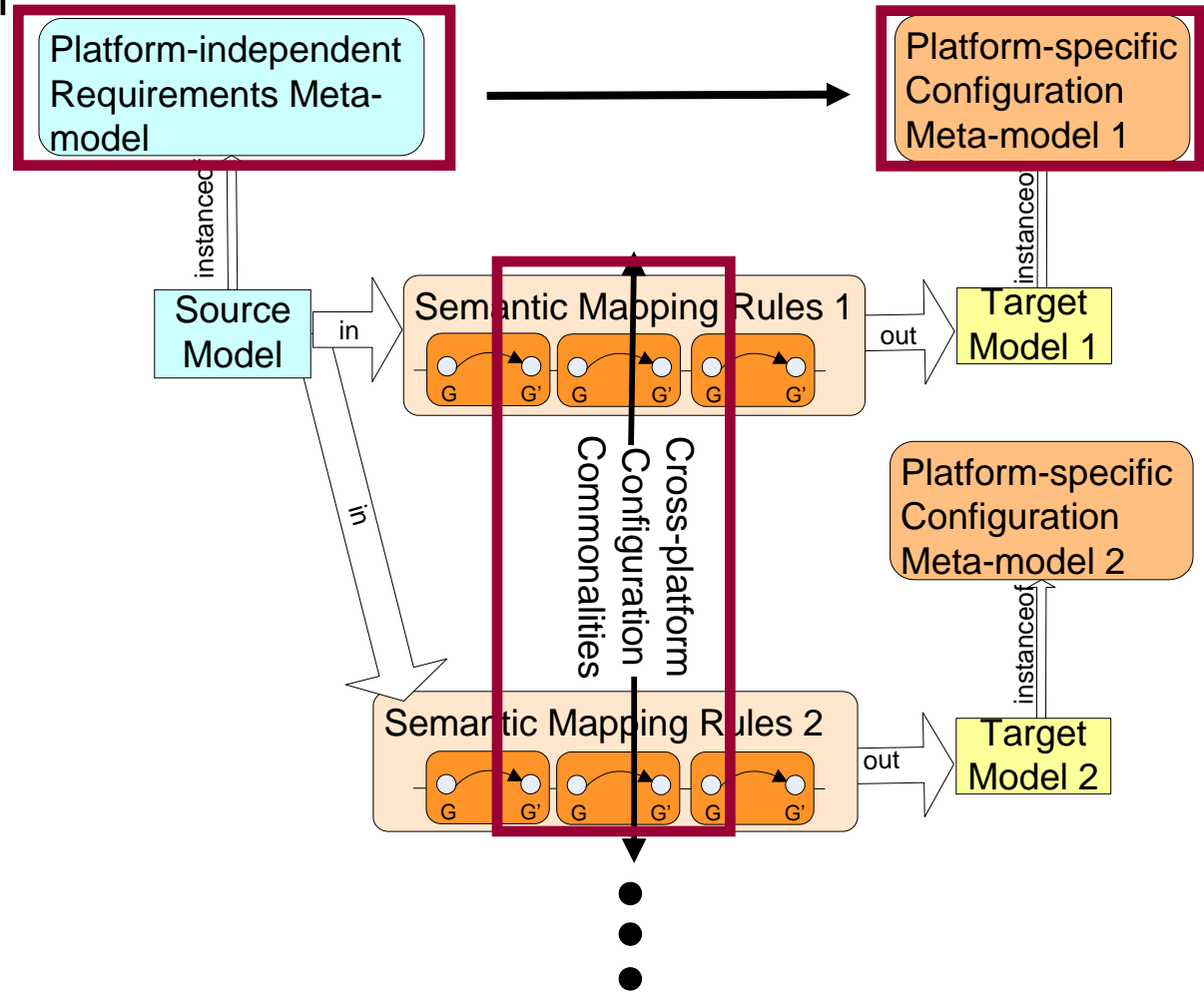


Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee



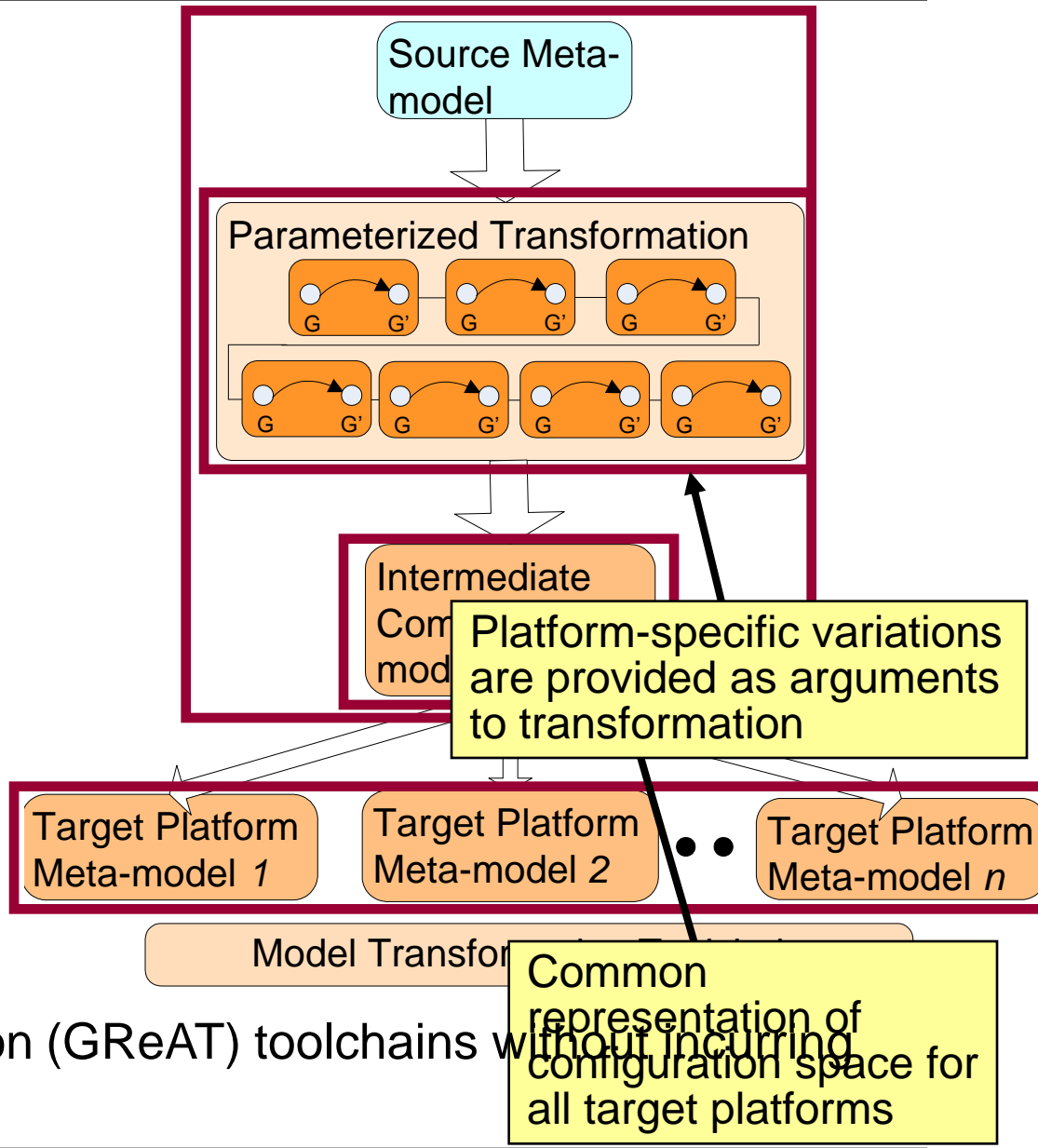
Context

- Middleware QoS configuration maps domain requirements to platform-specific options
- Commonality patterns present in QoS configuration stage of individual platforms
 - Opportunity exists to write general-purpose, reusable model transformations
- Existing transformation tools and techniques do not support development of *parameterized* model transformations



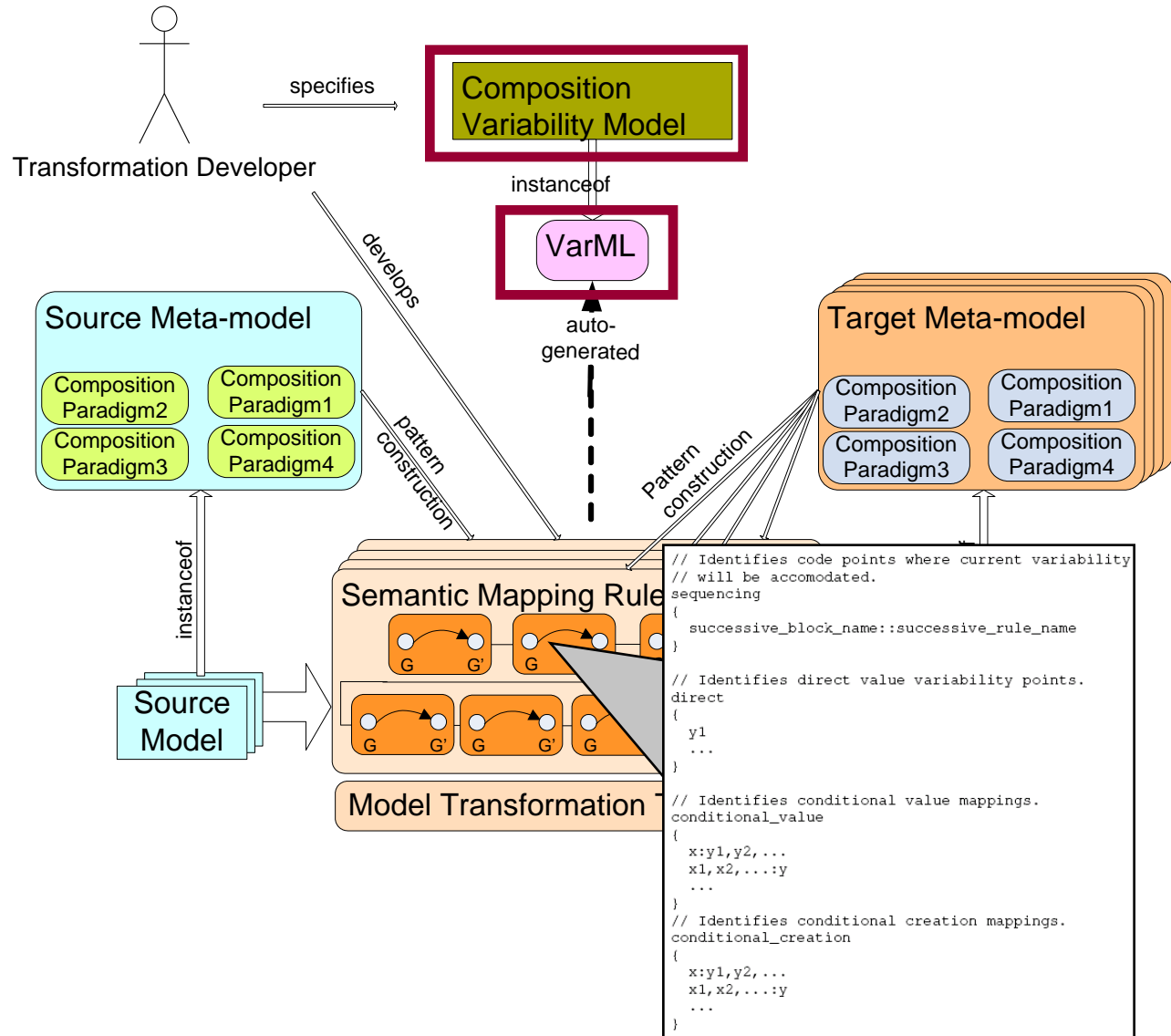
Parameterized Model Transformations

- Parameterized transformation performs general mapping of source (requirements) meta-model to target (configurations) meta-model
 - Common meta-model captures target platforms configurations in a QoS dimension
- Variations in QoS configuration (of individual platforms) are *decoupled* from transformation
 - Promotes reuse of same transformation for more than one platforms QoS mapping
- Extends existing modeling (GME) and model transformation (GReAT) toolchains without incurring additional overhead



Parameterized Model Transformations

- Transformation Parameterization for managing variation in rules composition:
 - Constraints notation blocks in transformation rules to identify points of variability
 - Auto-generated transformation instance-specific modeling language for specifying variabilities
 - Accommodating these variabilities in transformation



Identifying Composition Variability Points

- Constraint notation blocks can be used to identify the following variabilities:
 - *Direct Assignments* of target language objects
 - *Conditional Mappings* of target language objects from some source objects
- *Sequencing block* specifies locations in transformation project where variability would be accommodated
- Constraints notation block is opaque to transformation engine and does not interfere with its translation logic

```
// Identifies code points where current variability
// will be accommodated.
sequencing
{
    successive_block_name::successive_rule_name
}
```

```
// Identifies direct value variability points.
direct
{
    y1
    ...
}
```

```
// Identifies conditional value mappings.
conditional_value
{
    x:y1,y2,...
    x1,x2,...:y
    ...
}
```

```
// Identifies conditional creation mappings.
conditional_creation
{
    x:y1,y2,...
    x1,x2,...:y
    ...
}
```

Constraint notation blocks are inserted as special comments in transformation project

Specifying Transformation Variabilities

- General purpose transformations auto-generate VarML from source, target languages and transformation variability points
 - VarML can be used to specify transformation instance-specific variabilities
- Variabilities are expressed as name-value pairs (direct assignments) or associations (conditional mappings)
- Variability is extracted out from transformation project

```

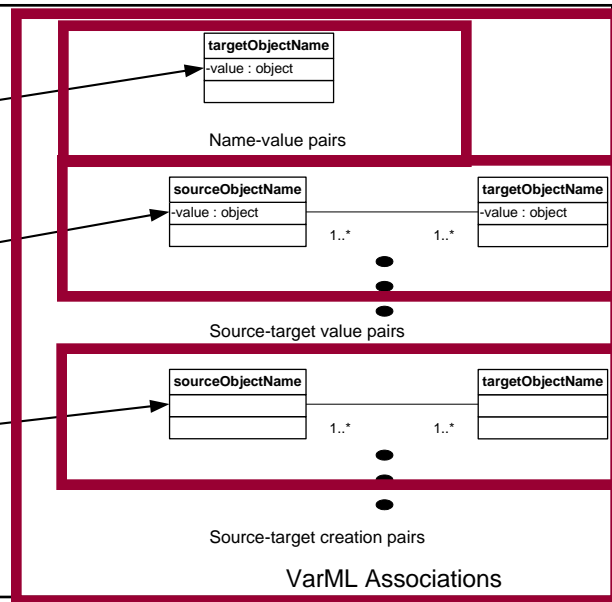
// Identifies code points where current variability
// will be accommodated.
sequencing
{
  successive_block_name::successive_rule_name
}

// Identifies direct value variability points.
direct
{
  y1
  ...
}

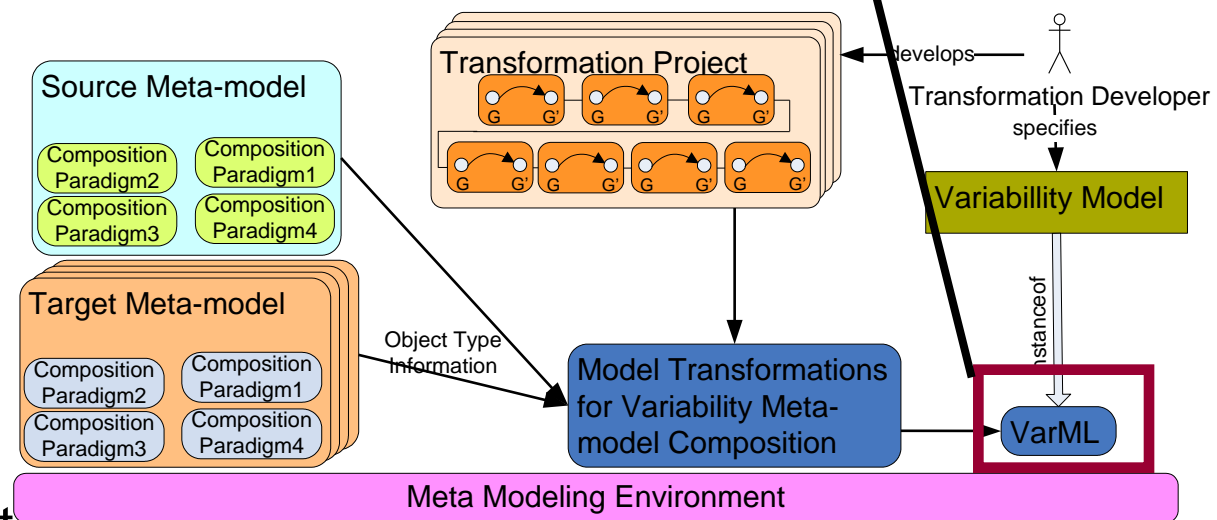
// Identifies conditional value mappings.
conditional_value
{
  x:y1,y2,...
  x1,x2,...:y
  ...
}

// Identifies conditional creation mappings.
conditional_creation
{
  x:y1,y2,...
  x1,x2,...:y
  ...
}
    
```

Transformation Algorithm Variability Points



VarML Associations



Accommodating Transformation Variabilities

- Finally, transformation variabilities are incorporated in transformation algorithms
 - VarML models used to construct new rules
 - Sequencing block used to determine the new rule location
- Successive changes in transformation do not change these rules

